(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 3 193 261 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
19.07.2017 Bulletin 2017/29

(51) Int Cl.:
G06F 17/22 *(2006.01)* ***G06F 17/24*** *(2006.01)*

(21) Application number: **17155254.0**

(22) Date of filing: **18.03.2014**

(84) Designated Contracting States:
AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR

(62) Document number(s) of the earlier application(s) in accordance with Art. 76 EPC:
14160413.2 / 2 921 970

(71) Applicant: **smartwork solutions GmbH**
81241 München (DE)

(72) Inventor: **Marchsreiter, Christian**
82205 Gilching (DE)

(74) Representative: **Bittner, Peter et al**
Peter Bittner und Partner
Seegarten 24
69190 Walldorf (DE)

Remarks:
This application was filed on 08.02.2017 as a divisional application to the application mentioned under INID code 62.

(54) METHOD AND SYSTEM FOR EDITING VIRTUAL DOCUMENTS

(57) A computer system (100), computer implemented method and computer program product for editing virtual documents. The computer system includes a storage component (110) configured to store a plurality of fragments (1 to 6) associated with a virtual document (190) that has a logical structure (140) wherein each fragment (1 to n) is stored separately from the other fragments. Further, a processing component (120) of the system is configured to assemble the virtual document (190) by retrieving the plurality of associated fragments (1 to 6) and ordering the plurality of associated fragments according to the logical structure (140). An editor component (130) of the system is configured to present a visualization (290) of the virtual document to a user (10) for editing and to receive editing commands from the user wherein a specific editing command (210) is configured to modify a specific fragment of the virtual document in the storage component (110).

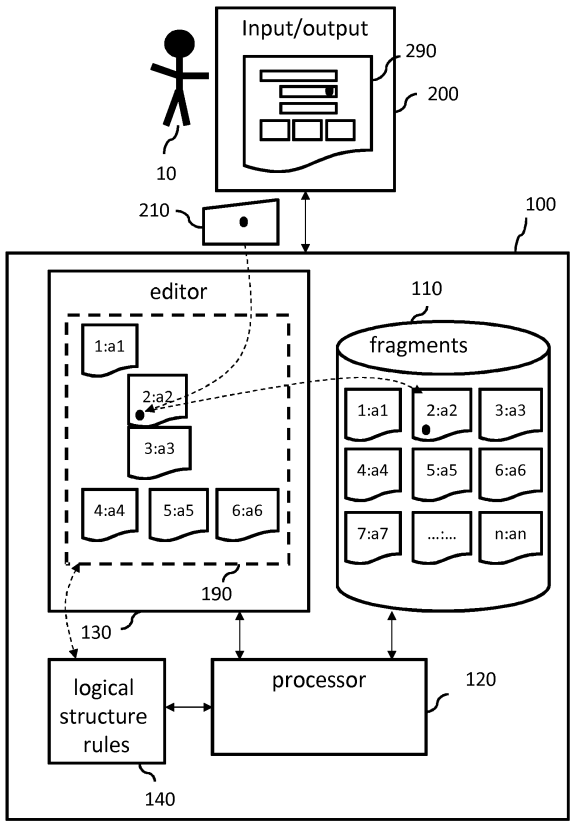


FIG. 1

EP 3 193 261 A1

Description

Technical Field

[0001] The present invention generally relates to electronic data processing, and more particularly, relates to methods, computer program products and systems for document editing.

Background

[0002] Standard editing tools provide tracking modes for tracking a part of the change history of a document. For example, this may be used to facilitate collaborative editing of a document. Track changes can show if something was added to or deleted from a document. Also an identification of the person responsible for the changes is typically stored with the track changes. However, in scenarios where many changes are applied by different users to the same document, often it becomes impossible to track the complete change history. Further, the change history of the document is typically visible for all users editing the document which may not be a preferred option when working on documents having a confidential character. Further, with many changes applied to a large document it becomes more and more difficult for a particular user to identify relevant changes quickly.

Summary

[0003] Therefore, there is a need to improve document editing systems with regards to their flexibility of handling large and complex documents and with regards to the accountability regarding changes and decisions to the document. A computer system, a computer implemented method and a computer program product as disclosed in the independent claims facilitates the flexible creation and change of documents which can be large and complex in structure.

[0004] The computer system for editing virtual documents has a storage component which can store a plurality of fragments associated with a virtual document. The virtual document that has a logical structure and each fragment is stored separately from the other fragments. The term virtual document as used hereinafter means that the document does not need to be stored or persisted in its entirety for editing purposes. It is dynamically composed or assembled from its fragments at runtime and visualized to a user. The visualization gives the user the impression to edit a complete document. However, in fact the visualization is only a view of the underlying fragments of the document.

[0005] A fragment may correspond to a headline of a document, a picture, a normal text portion and so on. Basically any piece of the virtual document which is in a way self-contained can be a fragment of the document. The logical structure of the virtual documents corresponds to a set of rules which is configured to evaluate the individual fragments in such a way that it becomes possible for the editing system to assemble all fragments of the virtual document in the correct order. The order relates to the arrangement or positioning of the fragments in space, for example, in top / down direction along a presentation medium such as a page or a display.

[0006] For this purpose, the computer system includes a processing component which is configured to assemble the virtual document by retrieving the plurality of associated fragments (e.g., from a fragments database in the storage component) and to order the plurality of associated fragments according to the logical structure of the virtual documents. Further the computer system has an editor component to present a visualization of the virtual document to a user for editing. The editing component can further receive editing commands from the user wherein such editing commands can be configured to modify a specific portion of the virtual document which corresponds to a specific fragment of the virtual document in the storage component. An editing command to modify the document may be directed to create or delete/deactivate a specific fragment, insert information into the specific fragment, or delete information from the specific fragment. It is to be noted that any change of existing content of a fragment is composed from deleting and inserting information. As explained further down, a fragment once created is not really deleted anymore. However, it may be deactivated and moved into some kind of archive from which it may be reactivated anytime if the respective editing user is authorized to do so.

[0007] The use of the logical structure for the virtual document allows the computer system to perform consistency checks for guaranteeing that no modifications are applied to the virtual document which would risk the structural integrity and the logical consistency of the virtual document.

[0008] For this purpose, the computer system may load instructions of a computer program which, when executed by one or more processors of the computer system, execute the following method.

[0009] The computer system receives an editing command from a user wherein the editing command is configured to modify the virtual document. The virtual document has said logical structure and displayed to the user on any appropriate graphical output means (e.g., monitor, touch screen, etc.). The editing commands may be entered by the user using any appropriate input means such as a keyboard, a mouse device, a touch display, etc.

[0010] The system then determines a fragment of the virtual document which has an object type that complies with

the received editing command. For example, in the user indicates that a new image is to be inserted into the virtual document the system creates a new fragment of the type image. If a new paragraph is to be inserted a corresponding fragment of type text may be created. If a headline is to be changed from unnumbered to numbered headline the respective text fragment is determined in the plurality of already existing fragments.

[0011] Further, the computer system determines a fragment dependency for the determined fragment if the determined fragment has a preceding fragment in the virtual document as a parent fragment. For example, if the editing command indicates to insert a standard text paragraph that relates to a previously created headline the system recognizes this dependency and can mark the newly created standard text fragment as a child of the previously created headline fragment by storing the parent's ID in the standard text fragment. Further, the fragment dependency for the determined fragment may be based on an indenture level associated with the determined fragment. Indenture level as used hereinafter relates to the various technical structuring levels of the virtual document. It is not meant to be limiting with regards to any need for a visual indenting in the presentation of the virtual document. Rather, it relates to any level associated with any hierarchy criterion which may be used for defining the structure of the virtual document.

[0012] The system checks the consistency of the fragment dependency with the logical structure of the virtual document. That is, the logical structure provides all rules for defining allowed relationships within the document. If the editing command is directed to a change of the virtual document infringes the logical structure of the virtual document then the consistency check would fail. If the consistency check fails, the system rejects the modification of the virtual document in accordance with the received editing command. That is, the system can automatically guarantee that no amendments are applied to the document which would go against the predefined logical structure. As a consequence, a user can never destroy the structure of the virtual document in a way that it makes no sense anymore.

[0013] The editor component therefore enables insertion, deletion or any movement of fragments (horizontal and vertical) but at the same time prevents destruction of the logical structure if a modification would result in, for example, illogical numbering or undesired formatting. By using indenture levels in one embodiment, the editor can ensure the repositioning of existing fragments in the virtual document is only enabled for meaningful modifications. This may include that moving a particular fragment results in the movement of all child-fragments, too. This may also include that deleting a fragment results in the deletion of all child-fragments.

[0014] If the consistency check is successful, the system stores the determined fragment as a separate database object in a fragment database together with the fragment dependency. As a consequence, only fragments affected by a modification of the document need to be changed at the database level. This allows a high level of granularity when tracking changes. Further, only the changed parts need to be retrieved from the fragment database when another user wants to review the virtual document including previous changes. This delta mechanism also saves processor performance when storing changes because only relatively small fragments are written to the database instead of the full document (as with standard document formats).

[0015] The system then visualizes the virtual document based on the current status of the fragment database and the logical structure of the virtual document. That is, only fragments affected by successful modification editing commands need to be updated at the visualization layer. Again this saves processing resources and eventually bandwidth when users edit the virtual document remotely. The logical structure allows reducing complexity of document creation because hierarchical structuring of the virtual document can be combined with global formatting rules which automatically ensures logical consistency and appropriate formatting of the respective fragments.

[0016] In one embodiment, the system can support filtering fragments of the virtual document based on at least one attribute of the fragments in the fragment database wherein at least one attribute controls the visibility of the at least one attribute for a particular user. This provides a high degree of flexibility with regards to managing the virtual document in multi-user scenarios to provide individual views on the document for each individual user. Filtering functions can also be used for generating dashboards or other forms of reports providing aggregate views on a virtual document. Management summaries regarding the change history or the content may be generated by simply executing predefined filters with regards to the relevant attributes (e.g., show only fragments which are critical and have high priority).

[0017] In one embodiment, if the consistency check is successful, the system inserts into the determined fragment version information which is associated with the modification applied in response to the editing command. The version information can be used for tracking the modification wherein the insertion is under the exclusive control of the computer system. In other words, a user cannot legally prevent the system from tracking any changes applied to the virtual document. Storing version information at the level of fragments allows fine granular versioning for every piece of the virtual document stored as a fragment. Version control mechanisms are therefore not applied to the entire document as in prior art solutions but rather to the various components from which the virtual document is dynamically composed. The system can then adjust the visual appearance of modified content in the presentation of the virtual document dependent on a modification type associated with the modification in response to the editing command. That is, each fragment includes its entire change history which can now be visualized to the user by appropriate visualization techniques as further detailed in the detailed description part.

[0018] In one embodiment, the version information can include metadata with a version number and a user identifier

of the user. The metadata may enclose content of the specific fragment which reflects the modification associated with the inserted version information. The metadata may be computer-readable or computer-parsable meta-data such as tags or any content descriptor that is delimited from the content and not to be rendered to the user. However, the metadata may affect rendering of the fragment as disclosed in the detailed description.

[0019] In one embodiment, the storage component may further store for a particular fragment a level of awareness for the user when the user accesses the particular fragment. The level of awareness may include a user identifier of the user and a current version of the particular fragment. The level of awareness reflects the user's knowledge during the user's latest view of the virtual document. For example, when the user returns to the virtual document after other users have modified certain fragments, the system can automatically apply a visualization filter which only presents changes to the user that occurred after his last view by comparing version information associated with the changes to the user's level of awareness.

[0020] Further aspects of the invention will be realized and attained by means of the elements and combinations particularly depicted in the appended claims. It is to be understood that both, the foregoing general description and the following detailed description are exemplary and explanatory only and are not restrictive of the invention as described.

Brief Description of the Drawings

[0021]

FIG. 1 is a simplified block diagram of a computer system for editing virtual documents according to one embodiment of the invention;
FIG. 2 illustrates fragments associated with portions of a virtual document;
FIGS. 3A and 3B illustrate visualization examples for various modifications according to optional embodiments of the invention;
FIG. 4 is an example of fragment versioning according to an embodiment of the invention;
FIG. 5 is an example of level of awareness visualization according to an embodiment of the invention;
FIG. 6 is an example of history drill down for a fragment according to an embodiment of the invention;
FIG. 7 is a simplified flowchart of a computer implemented method for editing virtual documents according to an embodiment of the invention;
FIG. 8 shows an example of the logical structure of a virtual document; and
FIG. 9 illustrates how fragments are stored in a fragment database.

Detailed Description

[0022] FIG. 1 is a simplified block diagram of a computer system 100 for editing virtual documents 190 according to one embodiment of the invention. The computer system 100 is communicatively coupled with an input/output (IO) device 200. The IO device 200 can also be an integral part of the computer system 100 or it may remotely be connected to the computer system through any standard communication interface. The IO device 200 can be used by a user 10 as a front end to edit a virtual document 190 by presenting a visualization 290 of the virtual document 190 to the user 10. As mentioned earlier, a virtual document is a document which is not persisted as a whole but which is dynamically assembled from fragments (1 to n) which are stored in a storage component 110 of the computer system 100. The storage component 110 can be a conventional database, a file system or any other appropriate storage structure where each fragment can be persistently stored separately from the other fragments. The virtual document is associated with a logical structure 140 associating the respective fragments with the virtual document 190. The association may be implemented by references in the logical structure 140 to the respective fragment 1 to 6. Such references may directly point to the identifiers of the respective fragments 1 to 6. The references may also be based on annotations a1 to a2 of the respective fragments. The logical structure 140 includes a description of how the various fragments are part of the virtual document. This may include a hierarchy definition of fragments corresponding to various sections of the document. Any other document specific structuring element like paragraphs, headlines, titles, enumeration elements, etc. can be defined regarding their size and location.

[0023] It is to be noted that at no point in time there is a need to persist the virtual document in the computer system as a single document for the purpose of editing the virtual document (snapshots of the virtual document may be created for other purposes but are not needed for the editing of the virtual document). Rather, the virtual document 190 is assembled at runtime (when the virtual document is presented to the user) by a processing component 120 of the computer system. When the user 10 requests the visualization 290 of the virtual document 190, the processing component retrieves the respective associated fragments 1 to 6 and orders the fragments according to the logical structure 140 associated with the virtual document 190. An editor component 130 of the system can render the assembled fragments according to the logical structure 140 resulting in the requested visualization 290 which is sent to the IO device (e.g., a

client computer with an output component such as a display component). Thereby, the content of the fragments is presented to the user like corresponding document elements of a complete single document. More details are disclosed with FIG. 2.

[0024] The user 10 may now modify the virtual document 190 by interacting with the editor component 130 of the computer system through the IO device 200. For example, the user can indicate a modification to be applied to any document element shown in the visualization 290. In the example, by using input means of the IO device 200 (e.g., mouse, touch screen, keyboard, etc.) the user indicates a change (illustrated by black bullet) to the second document element. The IO device 290 sends the respective editing command 210 to the editor component to modify the corresponding fragment 2 of the virtual document 190 in the storage component 110.

[0025] The proposed system allows very flexible and granular management of tracking document modifications.

[0026] FIG. 2 illustrates fragments 1 to 8 associated with portions of a virtual document. The example of FIG. 2 illustrates the visualization of the virtual document including seven document portions: a title, a first headline, a first example paragraph, a second example paragraph, a second headline, a third example paragraph and an enumeration block with two enumeration elements. Such portions of the document are presented to the user as if they were part of a standard text processor document, such as for example, a MICROSOFT WORD document or any other standard word processor document or any other document type, like for example, presentation documents or publishing documents may be used as well. However, according to an embodiment of the present invention, each of the mentioned portions is associated with a corresponding fragment 1 to 8 which is stored separately in the storage component of the computer system. The visualization 290 is a view on the respective fragments which is assembled according to the logical structure describing the virtual document. Fragment numbers 1: to 8: in FIG. 2 are typically not shown to the user.

[0027] Each fragment 1 to 8 can be associated with an object type. Examples, for object types are: text object, picture/image object, table object, etc. Any object type which is known from traditional text processing tools or graphical editing tools can be supported. Any fragment can have an object type different from the other fragments. Multiple fragments can have the same object type.

[0028] The logical structure may define logical dependencies and/or content dependencies between the fragments 1 to 8 within the virtual document. For example, fragments may be positioned in the virtual document in an arbitrary number of indenture levels. Indenture level as used hereinafter not only relates to the structuring of text documents. Also the structure of a graphic oriented sheet (e.g., a presentation sheet) may have a logical structure (e.g., groupings, drawing levels, etc.,) which can be expressed through the use of indenture levels. In the figure, the indenture level is symbolized by the horizontal distance from the text portions to left margin. For example, the first fragment 1 is a title which is associated with the highest indenture level (shortest distance). The second fragment 2 is a first headline which also is associated with the highest indenture level. The logical structure of the document includes a set of rules which defines where a fragment is to be positioned in the virtual document for visualization. For example, such a rule may define that a title at the first indenture level is always preceding a numbered headline at the same indenture level. Therefore, the system knows in which order the fragments 1 and to need to be arranged. The logical structure is complemented by definitions of parent-child relationships between fragments and corresponding indenture levels for fragments which are associated with subordinate indenture levels. This allows the user to define an error-free and logically correct structure for the fragments within the virtual document. For example, a particular fragment associated with a particular sub-level being a child of a superordinate parent level can only be created if the parent level actually exists. In other words, the parent-child relationships defined by the logical structure provide a self-consistency check with regards to the integrity of the document structure in view of predefined consistency rules in the logical structure rule set. In the example, the first numbered headline fragment of second indenture level (fragment 5: second headline) can only be created underneath the first indenture level numbered headline fragment (fragment 2: first headline). A user can only execute changes to the virtual document which result in fragments that do not violate the logical structure rules of the virtual document. For example, the fragment 5 (1.1 second headline) associated with the second indenture level of the virtual document can only be created by the user if the corresponding 1st indenture level exists. There can be multiple fragments at the same indenture level (e.g., fragments 3, 4 and 5). For example, fragment number 4 is at the same level as fragment number 5 but both are subsequent to fragment number 2. Therefore, the first headline (fragment 2) is at the first indenture level. The first example paragraph (fragment 3) and the second example paragraph (fragment 4) as well as the second headline (fragment 5) are at the second indenture level. Similar dependencies are defined for the fragments 6 to 8 which are associated with the third indenture level. As a consequence a parent-child relationship between two fragments may be based on the corresponding indenture level relation.. For example, the second numbered headline (fragment 5) is a child of the first numbered headline (fragment 2). The first example paragraph (fragment 3) is also a child of the first headline (fragment 2) and the relation is between the two corresponding indenture levels. The system knows that headline fragment 5 is to be positioned subsequent to the text paragraphs fragments 3, 4 because of a corresponding rule in the logical structure of the virtual document. In addition, a fragment may store further dependencies which reflect the relationship with other fragments. For example, a fragment may store positioning information which indicates where the fragment is to be positioned when the virtual document is assembled. Such positioning information can include the

predecessor of a fragment or the successor of the fragment. In any case the system is able to recognize that, for example, fragment 3 (first paragraph) precedes fragment 4 (second paragraph). If for example, a third paragraph is inserted between the first and second paragraphs the corresponding newly created fragment may include a reference to its preceding fragment 3 and the succeeding fragment 4 would change it reference to the newly inserted fragment as the preceding fragment.

[0029] Any dependencies being defined between two or more fragments while those fragments are being created or modified by the user are automatically stored and administrated by the system in the fragment database (e.g., storage component 110, cf. FIG. 1). Details of the virtual document creation workflow are disclosed in FIG. 7.

[0030] The editor component enables the user to create of modify a virtual document by creating/modifying and continuously administrating the respective fragments as database objects in the fragment database without any need of programming knowledge or code writing.

[0031] FIGs. 3A and 3B illustrate visualization examples 2001, 2001 for various modifications according to possible embodiments of the invention. When the computer system receives a specific editing command to modify a specific fragment such a modification command may be directed to creating a whole new fragment.

[0032] For example, turning briefly to the example of FIG. 2 and assuming that the user is just adding the first and second enumeration elements which may not have been present before, the editor component may trigger the creation of respective new fragments 7, 8 in the storage component. Further, the editor component may update the logical structure associated with the virtual document with the information that two additional elements (enumeration elements) have been added at the bottom location of the virtual document and that those additional elements are associated with the fragments 7, 8. In other words, if the user indicates a modification which extends the virtual document beyond its current structure (logical structure editing command) then the editor component triggers the creation of one or more corresponding fragments in the database extends the logical structure of the virtual document accordingly.

[0033] The modification command may further be directed to inserting information into the specific fragment. Again referring to the example of FIG. 2, inserting information may include adding one or more characters to any of the paragraphs of the virtual document. The editor component can then initiate the corresponding modification of the fragment in the storage component (e.g., database).

[0034] The modification command may further be directed to deleting information from the specific fragment. Again referring to the example of FIG. 2, deleting information may include deleting one or more characters from any of the paragraphs of the virtual document. The editor component can then initiate the corresponding modification of the fragment in the storage component (e.g., database).

[0035] FIG. 3A shows a table illustrating an example embodiment 2001 for visualizing modifications applied to fragments as well as for visualizing decisions applied to said modifications later on. Each modification has a modification type. Insert and delete modifications are of type change indicating that an effective change is applied to the content or structure of the virtual document. Accept and reject modifications are of type decision indicating that an applied change to the document can still be rejected or accepted by a user. Each modification is represented by a unique modification code (tag code column). The presentation column shows examples how the editor component can adjust the visual appearance of the modification content in the presentation of the virtual document dependent on the modification type for each modification. If an insert modification command is received (e.g., the user has inserted the word "example" into an existing paragraph), the modification can be tracked with the respective tag code INS at the fragment level. Any other appropriate tag code or tracking means may be used instead. For example, instead of tags binary codes may be used. The tag code is interpreted by the editor as an instruction to adjust the presentation of the inserted part according to the modification and type in the visualization of the virtual document associated with the modified fragment. In the example, an inserted string is underlined with a dotted line. In the example of FIG. 3B in an alternative embodiment the inserted character is underlined by a solid line. Any other suitable presentation may be chosen. For example, color codes can be used instead so that an inserted string is displayed in a color (e.g., green) different from the previously existing text (e.g., black). Other presentations, like for example different font sizes, font styles or animation effects like blinking may be used instead. In the embodiment 2001 the delete modification (type change) (e.g., triggered by a delete modification command to delete the word "example" from a paragraph) can be tracked, for example, with the tag code DEL, at the fragment level. The presentation for the delete modification is a dashed line striking through the deleted content. In the embodiment 2002 (cf. FIG. 3B), the presentation is a solid strike through line. Again, any other appropriate visualization can be chosen by a person skilled in the art.

[0036] Modifications of type change can be seen as intended modifications which still require final approval by the same or a further user. Such approvals are also tracked by the computer system at the fragment level, therefore modifying the fragment accordingly. In other words, an approval decision is a fragment modification on its own. Such approval modifications can have a corresponding type (decision). Such decisions will finally accept or reject an intended insertion or an intended deletion. While standard document editing tools only allow seeing if there is an intended modification or a decided modification, such tools typically fail to track detailed decision history of such modifications.

[0037] In the embodiment 2001 the accept insertion modification (tag code INSACC) is presented to the user by

underlining the inserted string with a solid line. That is, the dotted underline from the insert modification is changed to a solid underline when the insertion is accepted by a user. If the user however rejects the insertion (tag code INSREJ), another presentation is shown to the user. In the example, the dotted underline is kept but in addition a solid strikethrough line indicates that the insertion is rejected. In standard editing tools the final result would simply be a paragraph including the word "example" in the acceptance case and not including the word "example" in the rejection case. History tracking of the decisions leading to the final state would not be possible. This can be very disadvantageous in collaborative editing scenarios where many users create many insert and delete modifications. Embodiment 2002 shows an alternative presentation where decision modifications are presented with a frame around the modified element. An empty frame is used for accept decisions and a frame with a background pattern is used for reject decisions. Any other appropriate visualization, such as, for example, colored background or colored line styles may be used as well.

[0038] The same logic may be applied to the modifications related to the accept deletion decision and reject deletion decision as can be directly seen form the presentation column of the alternative embodiments 2001, 2002. Instead of using a binary logic with the accept decision and the reject decision, other logics can be applied as well, for example, a logic with further states: tentatively accept, definitively accept, tentatively reject, and definitively reject.

[0039] FIG. 4 is an example of fragment versioning according to an embodiment of the invention. In this embodiment, the modification associated with a particular modification command causes insertion of corresponding version information into the particular fragment for tracking the modification wherein the insertion is under the exclusive control of the computer system. The insertion of version information under the exclusive control makes the system inert against any attempt of a user to manipulate or even forge the change history of a fragment. Especially but not only in the context of collaborative editing the disclosed fragment versioning method provides an auditing proof method which is appropriate for the iterative creation of documents, such as for example, legal contracts, patent applications or any other document with a need to track change history, such as contacts. The table in FIG. 4 illustrates and example of stepwise modifications of a fragment indicated in the left column of the table and the respective fragment content including the version information by using history tags. The version information in a history tag includes a version number and a user identifier of the user causing the modification. Version information may further include a timestamp (date and time) indicating when the version information was created. For simplicity reasons timestamp are not included in the current example as they are not needed for the basic versioning functionality. The first modification relates to the insertion of the word "insertion" illustrated by the dotted underline. The respective fragment content is: "Original text plus an <INS, v=001, UID=145> insertion <END>." The inserted text is enclosed by a respective history tag pair: <INS, v=001, UID=145><END>. The opening tag includes the modification tag code INS, the version number v=001, and the user identifier of the modifying user UID=145. The inserted part "insertion" is enclosed by the start tag and it corresponding end tag <END>. The next modification is a deletion (dotted strike through) of the words "Old text" from the same fragment but through a different user (UID=063). The corresponding start tag including the modification tag code and the version information is: <DEL, v=002, UID=063>. The (to be) deleted portion "Old text" is delimited from the remaining content by the <END> tag. The users cannot influence the incrementing of the version number (v=002 for the second modification) with every modification of the fragment.

[0040] The next row of the table in FIG. 4 illustrates modifications of the decision type. The insertion caused by user (UID=145) is now accepted by a further user (UID=211). The acceptance of the insertion corresponds to a new modification which inserts new version information into the fragment. The corresponding start tag is inserted behind the earlier start tag of the original insert modification. The respective history tag <INSACC, v=003, UID=211> again includes the modification tag code INSACC, the new version number v=003 and the accepting user's UID=211. In the example, only one end tag is used to close both start tags. Alternatively, each start tag could be closed by a corresponding end tag. In a next step a still further user (UID=966) rejects the deletion of the (to be) deleted string "Old text". The presentation makes this immediately visible to any user looking at the visualization of the virtual document. In the example, the presentation of FIG. 3A is used. The fragment is extended by a further history tag <DELREJ, v=004, UID=966> reflecting the reject delete modification DELREJ of the still further user UID=966 under the version number v=004.

[0041] The combination of cascaded version information in the fragment and the possibility of visualizing modification history information allows a full tracking of all modification ever applied to the fragment in an audit proof manner. In conventional document editing solutions only to be made modifications can be visualized as long as they are not decided. Once decided (accepted or rejected) no change information is visible any more. To regain this information, typically prior art solutions force the user to perform compare operations on two different versions of the whole document. However, such a compare function only can provide the changes but not the information about who applied the changes to the document.

[0042] In other words, fragment versioning according to embodiments of the invention as disclosed enable tracking of the complete change and decision history of each fragment element (e.g., characters, strings, images, symbols, structure elements, etc.). Each modification of a fragment element results in stepwise extension of the respective version information (e.g., by adding a corresponding history tag describing the modification applied to the fragment element in the step corresponding to the respective version ID or version number). The granularity of fragment versioning as

disclosed allows differentiating between modifications of type change and decision, between users being responsible for the insertion of the history tags by the computer system, between different versions associated with the history tags, and optionally between timestamps associated with each history tag.

[0043] FIG. 5 is an example of level of awareness visualization according to an embodiment of the invention. Because each history tag includes the version ID/number that corresponds to the respective modification it is possible to present the visualization of each fragment for each user in such a way that a particular user immediately recognizes modifications applied to the document since the particular user viewed the visualization the last time. In other words, when the particular user views the visualization of the virtual document, and thereby the visualization of a particular fragment, the particular user obtains a corresponding level of awareness regarding the modification status of the fragment. Such level of awareness can be stored in the particular fragment for each user when viewing the particular fragment visualization. For example, the user level of awareness can be stored as a pair of the user's ID and the current version number of the fragment while the user is viewing the fragment. In the table of FIG. 5 the following example is shown. A first user UID=145 has made an insert modification with version v=001. A second user UID=063 has made a delete modification with version v=002. A third user UID=341 has made a further delete modification with version v=003. Each time when the users have viewed the fragment (which is always the case when making a modification) a corresponding user level of awareness is stored in the fragment.

[0044] When now the second user UID=063 views the fragment visualization again after the third user has made the delete modification with version v=003 the current user level of awareness for the second user is still at version v=002. That is, the second user is, at that time, not yet aware of the changes made by the third user. The editor component can take this into account when presenting the fragment content to the second user. For example, modification made after the version corresponding to the user's level of awareness can be highlighted in any appropriate way. In the example, the modification corresponding to version v=003 is shown in bold letters with an enlarged font size. Color coding or animation techniques may be used as well. When the user views the fragment content the user specific level of awareness is updated for the third user to version v=003 [UID=063, v=003]. The update (illustrated in italics) can be simply added to the existing records of user level awareness or it can replace the existing user specific awareness record [UID=063, v=002] to keep the size of a fragment as small as possible. When later the second user reviews the fragment content without any further modification being applied the presentation of modification associated with version v=003 will be presented according to the normal presentation rules for modification as laid out for example in FIGS. 3A, 3B. Because the user is already aware of all modifications applied to the respective fragment. This allows a user to immediately identify all modifications (changes or decisions) applied a virtual document since the last time the user has reviewed the document. In other words, the proposed technical fragment versioning method facilitates the editing of documents especially in a collaborative editing scenario, Further, collaborative editing becomes more transparent and less error prone when applying fragment versioning according to embodiments of the invention as disclosed. A further advantage of the fragment versioning method is the possibility to quickly guide a user to those parts of the virtual document where actually modifications have been applied since the user's latest access to the document. As the virtual document is assembled from fragments the editor component may apply a filter to the visualization of the fragments based on the user level awareness records of each fragment. For example, only fragments where modifications were applied since the user's last access may be presented. This can substantially reduce the size of the presented visualization. For example, in collaborative editing of large documents haven many pages only one paragraph may have been modified since the user's last access. Applying such a filter could result in a presentation of the fragment corresponding to the changed paragraph only. Alternatively, all fragments may be presented but fragments with new modifications may be highlighted in any appropriate way.

[0045] FIG. 6 is an example of a history drill down for a fragment according to an embodiment of the invention. A further advantageous feature of fragment versioning according to the invention is the possibility to instantly receive a complete drill down of the modification history for each element of a fragment. In the example of FIG. 6. Because the system tracks all modifications for each fragment element within the fragment, a user has the possibility to review the complete modification history of each fragment. The previously disclosed visualization features allow visualizing a combined change and decision history with regards to the latest decision modification of the respective fragment element. In the example of FIG. 6, four different users have applied modifications to the word "example". The first user inserted the word, the second user accepted the insertion, the third user deleted the word and the fourth user accepted the deletion. If now the first user is reviewing the fragment content again, the editor may not only show that the word was deleted and the deletion was accepted. This can be achieved by the general visualization rules as shown in FIGS. 3A, 3B. Moreover, the editor component can present an additional indicator 601 (for example, an information symbol) next to the modified element which indicates to the user that a modification history drill down is available for this fragment element. The user may now select the drill down indicator 601 causing the system to provide a history list 602 with all modifications that have been applied to the fragment element. Alternatively, the history list 602 may only include the track of unknown modifications, that is, modifications which were applied by other users. Alternatively, the history list 602 may only include the track of modifications since the last interaction of the user with the document.

[0046] FIG. 7 is a simplified flowchart of a computer implemented method 1000 executed by a computer system according to an embodiment of the invention. The computer implemented method 1000 enables a user to modify a virtual document using the editor component of the computer system and to generate or adjust fragments in a fragment database wherein the fragments are used to dynamically compose the virtual document at runtime.

[0047] The user can use the editor component to visualize and manipulate the virtual document. Thereby, the editor component may support any editing function known from conventional text or graphic editors. For example, an editing command can be received 1100 from the user through standard I/O means of the computer system (e.g., keyboard, mouse, touch display, etc.). The editing command may be directed to the modification of the currently displayed virtual document or it may relate to the creation of a new virtual document. For example, the user may push a particular key (e.g., RETURN, INS, DEL, etc.) or perform a respective menu function using a mouse device to indicate a desired modification of the virtual document to the computer system.

[0048] Thereby, the user may specify the object type of a fragment (e.g., text, image, table, etc.) which relates to the object to be modified. That is, documents may be composed from fragments corresponding to a text section, an image section, a table section, etc. Fragments may further have additional properties specifying further details of the fragment. For example, a text fragment may have the further property "numbered headline". A picture fragment may have the further properties "with flow text" or "in separate column". Such further properties can be evaluated by the logical structure rules to determine the positioning of the respective fragment in the virtual document. In case a new object is to be created in response to the editing command the computer system determines 1200 a new fragment according to the specified object type which corresponds to the to-be-created element. In case an existing object is to be modified the systems determines 1200 the corresponding affected fragment in the fragment database of the system.

[0049] The system further determines 1300 fragment dependencies reflecting the logical structure of the virtual document. In case of creating a new fragment the parent-child relationship with at least one previously created fragment is determined. More details regarding the logical structure information are disclosed in the context of FIGS. 8 and 9. For example, the user may determine the indenture level at which the to-be-created fragment will be positioned when a new fragment is to be created. This positioning may be changed at any time after the creation of the fragment. In this case the already existing fragment may be moved around in the virtual document which may lead to a change in the dependencies. That is, the already existing fragment may receive a new parent fragment when being moved to another position.

[0050] However, the system can always check 1400 the consistency with the logical structure rules of the virtual document. For example, indenture levels of an existing fragment may be changed as long as the logical consistency of the virtual document is not violated. In other words, the editor component can reject any modification (e.g., moving) of a fragment which would destroy the logical consistency of the virtual document. For example, a fragment belonging to indenture level 2 can only be moved to indenture level 3 if, after the modification there is still a fragment at indenture level 2 which precedes the moved fragment and can become the parent of the moved fragment. If only indenture level 1 and 2 fragments exist the system can prevent a movement of such fragments to indenture level 4 as no valid parent would be available for the modified fragment(s).

[0051] In other words, if the consistency check 1500 results in an inconsistent structure of the virtual document (NO) the modification request of the user is rejected 1600 by the system. On the other hand, if the consistency check 1500 results in a consistent structure of the virtual document the modified (created or changed) fragment is stored 1700 as a separate database object with its own object ID in the fragments database. This includes the storing of all determined fragment dependencies. That is, the parent-child relations defining the structure of the virtual document are stored together with the fragment content which allows flexible handling and fast composition of the virtual document based on the individual fragment database objects.

[0052] Once the editing command is completely processed by the editing system the virtual document is visualized according to the current status of the fragments database. That is, in case the modification was rejected by the system the virtual document is visualized as before receiving the editing modification command because no fragment changes were persisted in the fragment data base. However, if the determined fragment is stored together with its fragment dependencies, the stored changes are taken into account when visualizing the virtual document. In other words, the modification becomes immediately visible to the user after the respective fragment changes have been stored in the fragment database.

[0053] FIGS. 8 and 9 show an example of the logical structure of a virtual document and how the respective fragments are stored in the fragment database. In FIG. 8 each frame corresponds to a fragment of the virtual document 190. The first (upper) frame includes a headline of a first indenture level (root level) to be shown in the table of contents without numbering.

[0054] The corresponding fragment database object is shown in FIG. 9 as the database object with the ID=001. FIG. 9 shows an exemplary table of the fragment database 110. Each fragment is a record of the table with its own database object number (ID), a parent field (belongs to) for storing fragment dependencies, a type field (type) for indicating the type of the fragment, an indenture level field (ind. level) for assigning an indenture level to the fragment, a numbering indicator field (is headline <> numbering) for indicating if a headline should appear with numbering, and a content field

(content) for storing the content of the fragment. The content field also includes the versioning information for the fragment. A person skilled in the art may add further properties to a fragment database record and can select a different notation of fields of the fragment table and their field contents to achieve the same purpose. In the example, the fragment with ID=001 has no parent which indicates that it is a root fragment of the virtual document. It is of type text at the first indenture level and has the further property that it is a headline without numbering which causes its content to appear without numbering in the visualization of the virtual document.

[0055] The second frame in FIG. 8 includes a numbered headline of the first indenture level. The corresponding fragment with ID=002 is shown in FIG. 9. This fragment is similar to the first fragment ID=001. However, it has the further property that it is a headline with numbering being responsible for the numbering "1." (cf. FIG. 8) in the visualization of the virtual document. Fragments ID=001 and 002 are both positioned at the same indenture level (root level). There is no parent-child relationship between them. The positioning of the fragments in the virtual document is determined based on the additional properties of the fragments which are evaluated by the logical structure rules accordingly. In the example the logical structure rule includes that a text object being a headline without numbering precedes a text object being a headline with numbering.

[0056] The third frame in FIG. 8 includes text in relation to the numbered headline in the second frame. The corresponding fragment ID=003 (cf. FIG. 9) is a text fragment which is at the second indenture level. The logical structure of the virtual document may have a rule which automatically creates a fragment for a standard text object subsequent to a headline at the indenture level which is subordinate to the indenture level of the headline. Therefore, fragment ID=003 is a child of fragment ID=002 at indenture level 2 which allows to position it correctly in the virtual document. The same is true for fragment ID=004 which is again a text object with the additional property of a numbered headline. The logical rule set may include a rule which defines such a headline as succeeding a standard text object.

[0057] The fifth to eighth frames in FIG. 8 are defining a sub-section of the previous fragment ID=004 which is illustrated by the respective indent. The corresponding fragments are defined as the data records from ID=005 to ID=08. The fragments associated with the indented sub-section of the virtual document all belong to indenture level 3. Fragments ID=002 and ID=007 correspond to headlines with and without numbering, respectively. Fragment ID=006 is of type image with a picture of a person as its content and fragment ID=008 is of type table with a spreadsheet as its content. This illustrates the flexibility of the fragment database regarding any arbitrary type of object which can easily be included in the virtual document. To summarize, the parent-child relations can be used to define the relationship between a particular fragment and the respective parent fragment at the superordinate indenture level. For complex documents this clearly relates a fragment to the correct place in the virtual document with regards to the parent node. The positioning inside the indenture level underneath the correct parent is then determined by using the logical structure rules of the virtual document in combination with the additional properties of the fragment. Further examples for additional properties of a fragment are tags, ratings or priority settings, changes to the content, sequence numbers, etc. The fragment database in FIG. 9 may further include a positioning field for each fragment. Such a positioning field can be used to store references to the predecessor and/or successor fragment(s) of the respective fragment. For example, a virtual document related to content with little structure information (e.g., a patent application) may have many subsequent fragments of the same type at the same indenture level and having the same additional properties (e.g., standard text paragraphs). In such case the logical structure rules alone may not be able to guarantee a correct positioning of each fragment within the virtual document. The positioning field can be used to define a unique positioning within the virtual document for each fragment. In one embodiment, the" belongs to" field may be used to store the predecessor and/or successor object ID of the respective fragment, thus creating a parent child relationship reflecting the exact positioning information. Note that in case a new fragment gets inserted between two existing fragments, the references to the preceding or succeeding fragment are added to the new fragment but need to be updated in either the preceding or succeeding fragment dependent on the type of relationship used by the positioning field.

[0058] For each indenture level and/or for each additional property of a fragment the editing system can define standard formats at a global level. Global level in this context means that central formatting information is available for any fragment. Examples of such global formatting information are: font, font size, font color, font style (e.g., bold, italics, underlined, etc.), visualization distance between subsequent fragment contents, indenture (from left or right), orientation, etc. The global formatting can be applied to each fragment based on the type of the fragment, its additional properties, its positioning in an indenture list, etc. For this purpose, rules which are similar to the logical structure rules may be used.

table - global formatting example

indenture level	property	global formatting
1	headline with numbering	font=Calibri, size=18, bold, color=black, numbering_style="1", etc.
1	headline without numbering	font = Calibri, size=20, bold, color =black, etc.

(continued)

indenture level	property	global formatting
2	normal text	font = Calibri, size =14, color = black, etc..
2	headline with numbering	font = Calibri, size =14, bold, color = black, numbering_style ="1.1." , etc..
2	headline without numbering	font = Calibri, size =14, italics, bold, color = black, etc.
3	normal text	font = Calibri, size =12, color = black, etc.

The above table "global formatting example" shows an example how global formatting can be configured. The example relates to the virtual document shown in FIG. 8 and the respective fragment definitions in the fragment database 110 of FIG. 9. Dependent on particular combinations of the fragment properties as defined in the fragment database 110 for the various fragments the system can automatically determine the corresponding format information from the global formatting column. For example, second fragment ID=002 is a text element and at the same time a headline with numbering which is assigned to the first indenture level. Therefore, the criteria of the second entry in the global formatting table are met by the second fragment and the visualization (cf. FIG. 8, second frame) is adjusted accordingly. That is, the content of the second fragment is visualized by using the respective global formatting information "font = Calibri, size=20, bold, color =black, etc." The seventh fragment ID=007 is a text element and at the same time a headline without numbering which is assigned to the third indenture level. Therefore, the criteria of the fifth entry in the global formatting table are met by the seventh fragment and the visualization (cf. FIG. 8, seventh frame) is adjusted accordingly. That is, the content of the seventh fragment is visualized by using the respective global formatting information "font = Calibri, size =14, italics, bold, color = black, etc.", and so on. The global formatting allows a consistent formatting of the document without any burden of formatting considerations at the fragment level.

Global formatting can also be used for automatic reformatting of fragments which are repositioned in the virtual document. The global format which corresponds to the new position is the applied to the modified fragment.

[0059] The global formatting configuration may always be overwritten by a user for any portion of a particular fragment. Further, a fragment in the fragment database may have further attributes which can be set, changed or deleted any time by the user while the virtual document is being created or modified. Examples for such further attributes are: priority (e.g., high, medium, low), content based assessment (e.g., good, neutral, bad), bookmark, follow-up and/or finished dates, user defined or pre-defined tags, conversations for any defined group of editing users (e.g., discussions, questions, tasks, etc.), links to other fragments of the virtual document or to URLs not being listed in the content area of the fragment, files being uploaded to the fragment while respective links are not listed in the content area of the fragment, etc. Such further attributes allow expanding the virtual document editor into a powerful collaborative editing tool which can be customized on demand according to the users' needs and the type of document.

[0060] The separate storage of each fragment as a database object in combination with the functions of the further attributes at the fragment level allow a fine granular control of the virtual document with regards to filtering the presentation content of the virtual document for individual users. This is achieved by setting visibility flags for one or more attributes with regards to individual users or user groups. For example, a content based assessment attribute (good, bad, etc.) may be flagged as visible for only for reviewers of the virtual document who are part of the same company. Users of another company cannot access this attribute content. For example, a user may define that only changes made by other users should be visible to him which allows getting a quick overview of the contributions of co-editing users. Such filter functions may be used individually by every user of the editing system to change the view on the virtual document for focusing on such attributes of the respective fragments which are of relevance to the individual user. The filter functions facilitate the localization and further processing of relevant fragments of the virtual document.

[0061] It may be useful to present aggregate views for selected filter criteria to a user. This may provide a simplified view on critical or important elements or fragments of a virtual dashboard. For example, it may be interesting to provide information about

- how many sections or fragments of the virtual document include undecided changes,
- how many sections have a certain valuation (e.g., red or green) and what is the respective priority (e.g., 0-3),
- what is the ration between green marked and red marked sections, etc.

[0062] Statistics on such aggregate data may be provided to the user in the form of graphical dashboards supporting all kinds of data visualization techniques, For example, bar charts, pie charts, 3D graphics, line charts or any other appropriate diagram type to visualize the respective aggregate data may be used.

[0063] Method steps of the invention can be performed by one or more programmable processors executing a computer program to perform functions of the invention by operating on input data and generating output. Method steps can also be performed by, and apparatus of the invention can be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application-specific integrated circuit).

[0064] Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computing device. Generally, a processor will receive instructions and data from a read-only memory or a random access memory or both. The essential elements of a computer are at least one processor for executing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto-optical disks, or optical disks. Such storage devices may also provisioned on demand and be accessible through the Internet (Cloud Computing). Information carriers suitable for embodying computer program instructions and data include all forms of non-volatile memory, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in special purpose logic circuitry.

[0065] To provide for interaction with a user, the invention can be implemented on a computer having a display device, e.g., a cathode ray tube (CRT) or liquid crystal display (LCD) monitor, for displaying information to the user and an input device such as a keyboard, touchscreen or touchpad, a pointing device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input.

[0066] The invention can be implemented in a computing system that includes a back-end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front-end component, e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation of the invention, or any combination of such back-end, middleware, or front-end components. Client computers can also be mobile devices, such as smartphones, tablet PCs or any other handheld computing device. The components of the system can be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network (LAN) and a wide area network (WAN), e.g., the Internet or wireless LAN or telecommunication networks.

[0067] The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.

Claims

1. A computer system (100) for editing virtual documents, comprising:

a storage component (110) configured to store a plurality of fragments (1 to n) associated with a virtual document (190) that has a logical structure (140) wherein each fragment (1 to n) is stored separately from the other fragments;
a processing component (120) configured to assemble the virtual document (190) by retrieving the plurality of associated fragments (1 to n) and ordering the plurality of associated fragments according to the logical structure (140);
an editor component (130) configured to present a visualization (290) of the virtual document to a user (10) for editing and to receive editing commands from the user wherein a specific editing command (210) is configured to modify a specific fragment of the virtual document in the storage component (110);

characterized in that
the modification associated with the specific editing command comprises insertion of corresponding version information into the specific fragment for tracking the modification wherein the insertion is under the exclusive control of the computer system, and the version information comprises metadata with a version number and a user identifier of the user, the metadata enclosing content of the specific fragment which reflects the modification associated with the inserted version information.

2. The computer system (100) of claim 1, wherein the specific editing command to modify the specific fragment is selected from any of the following modification commands:

create the specific fragment,
insert information into the specific fragment, or
delete information from the specific fragment.

3. The computer system (100) of claim 2 being further configured to create the specific fragment in response to an object creation editing command for extending the virtual document (190) according to the logical structure (140).

4. The computer system (100) of any of the preceding claims, wherein the modification has a modification type, and wherein the editor component is configured to adjust the visual appearance of the modification content in the presentation of the virtual document dependent on the modification type.

5. The computer system (100) of any of the preceding claims, wherein the storage component (110) is further configured to store for a particular fragment a last viewed version information for the user when the user accesses the particular fragment, the last viewed version information comprising a user identifier of the user and a current version of the particular fragment.

6. The computer system (100) of any of the preceding claims, wherein each fragment (1 to n) is assigned to an indenture level of the virtual document (290).

7. The computer system (100) of any of the preceding claims, wherein fragments which are different from root fragments of the virtual document (190) include a reference to a parent fragment.

8. A computer implemented method for fragment versioning when editing a virtual document (190), comprising:

storing a plurality of fragments (1 to n) associated with a virtual document (190) that has a logical structure (140) wherein each fragment (1 to n) is stored separately from the other fragments;
assembling the virtual document (190) by retrieving the plurality of associated fragments (1 to n) and ordering the plurality of associated fragments according to the logical structure (140);
receiving, in response to a visualization (290) of the virtual document, editing commands from a user wherein a specific editing command (210) is configured to modify a specific fragment of the virtual document in the storage component (110);
inserting corresponding version information into the specific fragment for tracking the modification wherein the insertion is under the exclusive control of the computer system, and the version information comprises metadata with a version number and a user identifier of the user, the metadata enclosing content of the specific fragment which reflects the modification associated with the inserted version information.

9. The method of claim 8, further comprising:

storing for a particular fragment a last viewed version information for the user when the user accesses the particular fragment, the last viewed version information comprising a user identifier of the user and a current version of the particular fragment.

10. The method of claim 8 or 9, wherein the version information further includes a timestamp indicating when the version information was created.

11. The method of any of the claims 8 to 10, wherein the insertion under the exclusive control of the computer system includes incrementing the version number with every modification of the fragment.

12. The method of any of the claims 8 to 11, wherein the modification is of a decision type indicating a decision on an earlier modification, and the specific fragment is extended by inserting version information with a corresponding history tag behind the history tag related to the earlier modification.

13. A computer program product that when loaded into a memory of a computing device and executed by at least one processor of the computing device executes the steps of the computer implemented method according to any one of the claims 8 to 12.

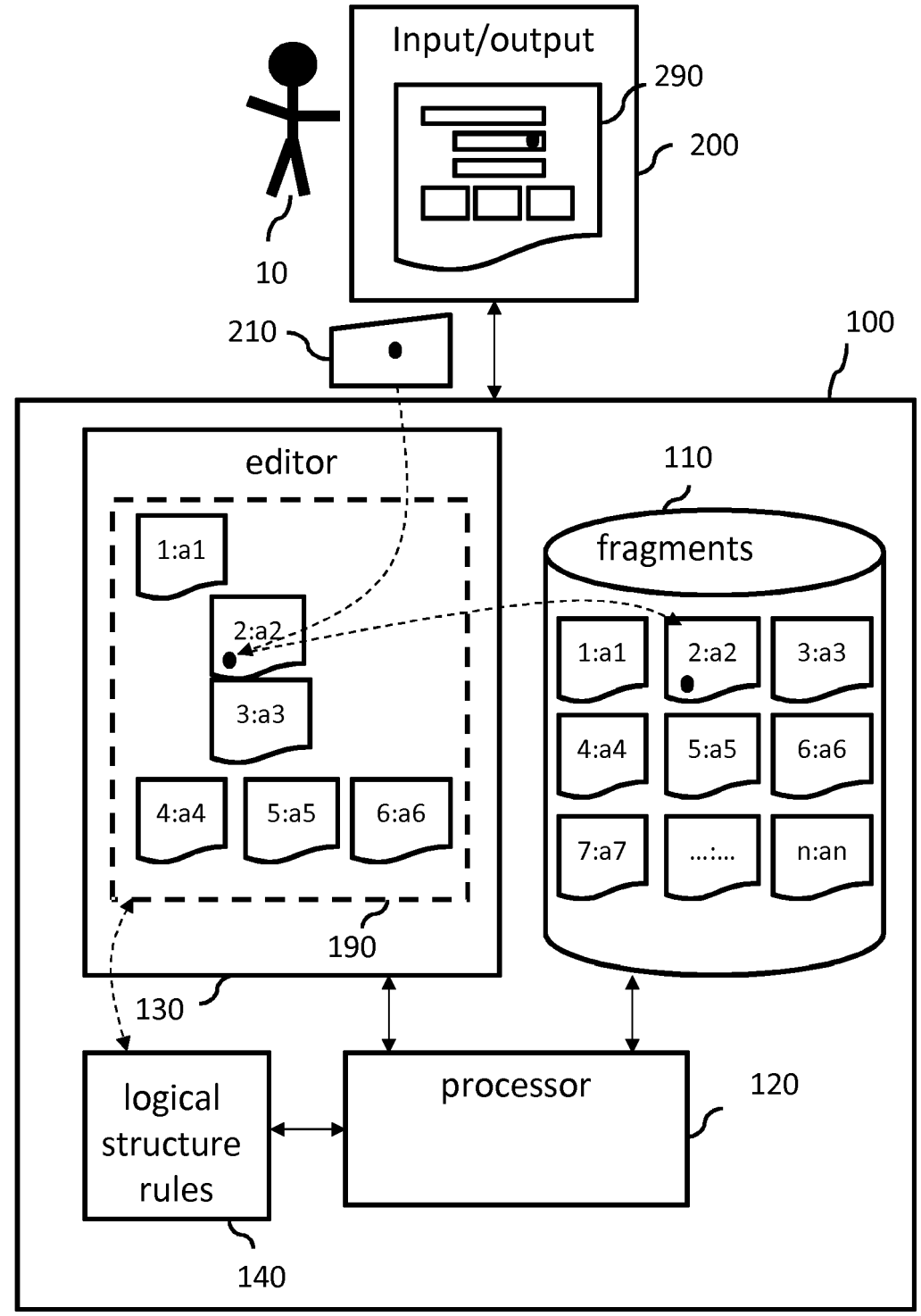

Input/output
290
200
10
210
100
editor
110
fragments
1:a1
2:a2
3:a3
4:a4
5:a5
6:a6
7:a7
...:...
n:an
190
130
logical structure rules
processor
120
140

FIG. 1

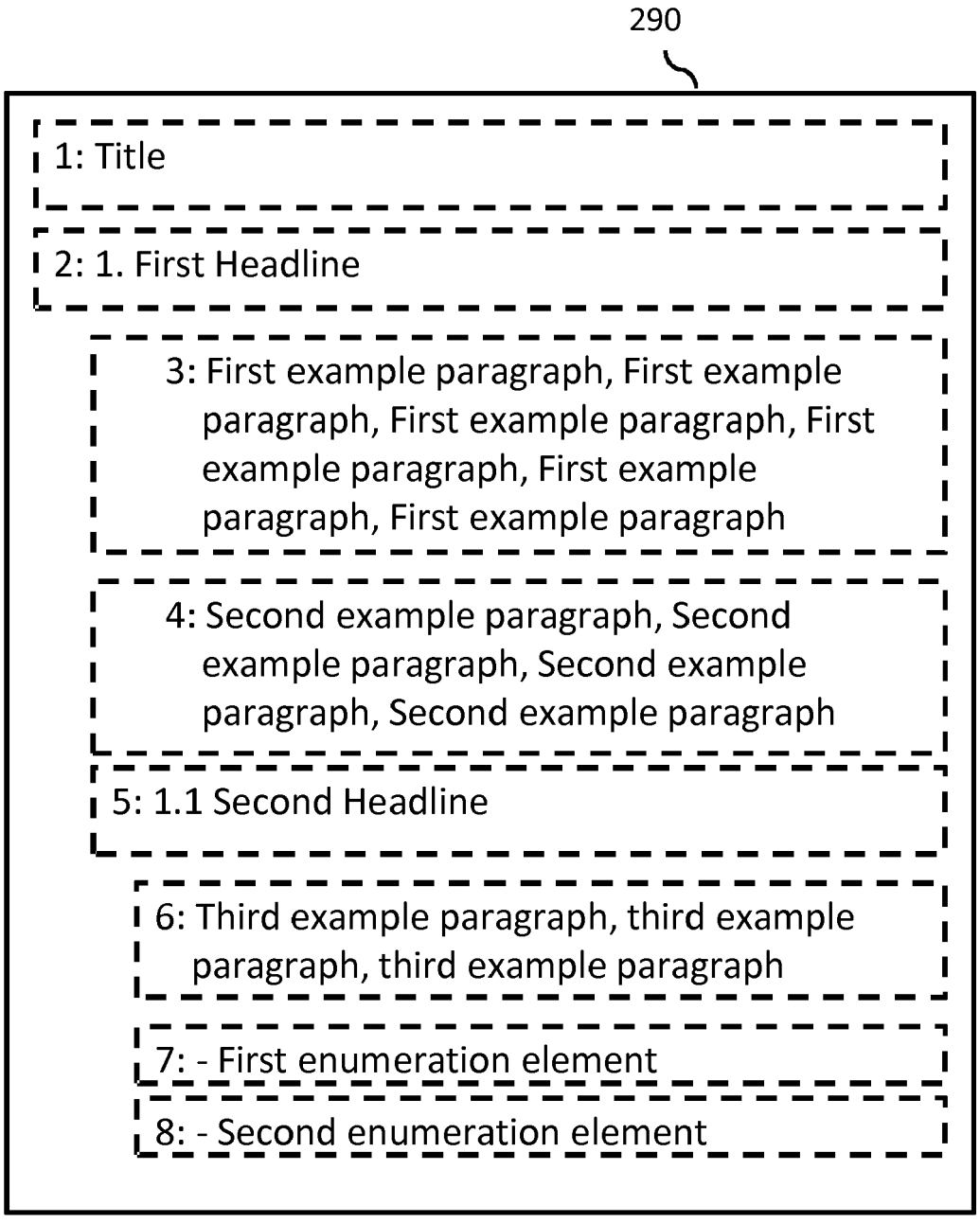

290
1: Title
2: 1. First Headline
3: First example paragraph, First example paragraph, First example paragraph, First example paragraph, First example paragraph, First example paragraph
4: Second example paragraph, Second example paragraph, Second example paragraph, Second example paragraph
5: 1.1 Second Headline
6: Third example paragraph, third example paragraph, third example paragraph
7: - First enumeration element
8: - Second enumeration element

FIG. 2

FIG. 3A

2001

modification	type	tag code	presentation
insert	change	INS	example
delete	change	DEL	~~example~~
accept insertion	decision	INSACC	example
reject insertion	decision	INSREJ	~~example~~
accept deletion	decision	DELACC	~~example~~
reject deletion	decision	DELREJ	~~example~~

FIG. 3B

2002

modification	type	tag code	presentation
insert	change	INS	example
delete	change	DEL	~~example~~
accept insertion	decision	INSACC	example
reject insertion	decision	INSREJ	example
accept deletion	decision	DELACC	~~example~~
reject deletion	decision	DELREJ	~~example~~

modification	fragment text with tags
Original text plus an insertion. ~~Old text~~ to be deleted.	Original text plus an <INS, v=001, UID=145> insertion <END>. <DEL, v=002, UID=063> Old text <END> to be deleted.
Original text plus an insertion. ~~Old text~~ to be deleted.	Original text plus an <INS, v=001, UID=145> <INSACC, v=003, UID=211> insertion <END>. <DEL, v=002, UID=063> <DELREJ, v=004, UID=966> Old text <END> to be deleted.

FIG. 4

modification	fragment text with tags
Original text plus an insertion. ~~Old text~~ **~~to be deleted~~**.	Original text plus an <INS, v=001, UID=145> insertion <END>. <DEL, v=002, UID=063> Old text <END> <DEL, v=003, UID=045> to be deleted <END> user level of awareness: ... [UID=145, v=001] [UID=063, v=002] [UID=045, v=003] *[UID=063, v=003]*

FIG. 5

modification	fragment text with tags
This is an ~~example~~ ⓘ.	This is an <INS, v=001, UID=145> <INSACC, v=002, UID=956> <DEL, v=003, UID=563> <DELACC, v=004, UID=341> example <END>.

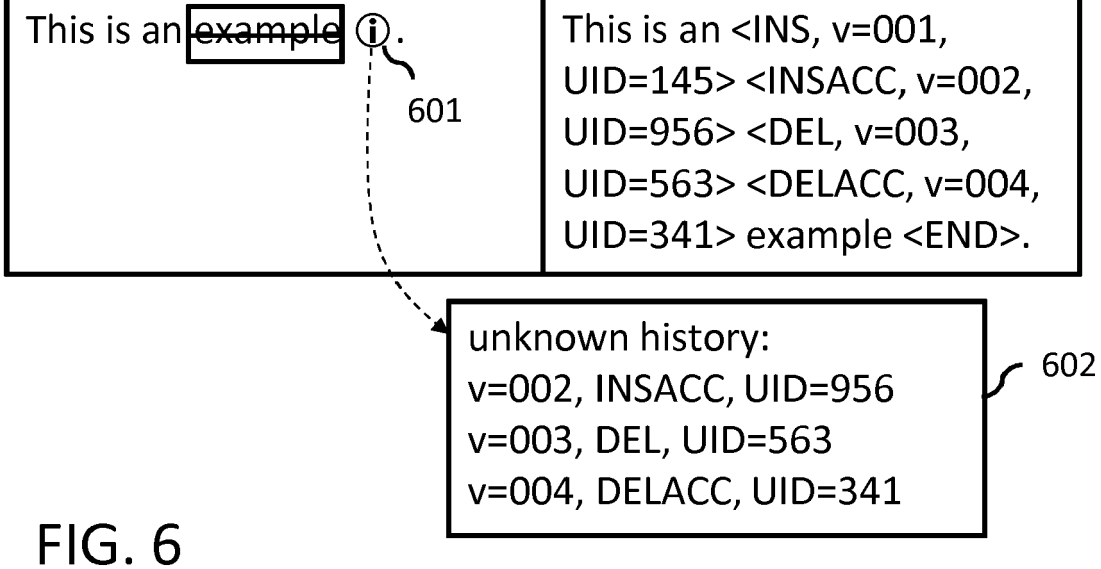


FIG. 6

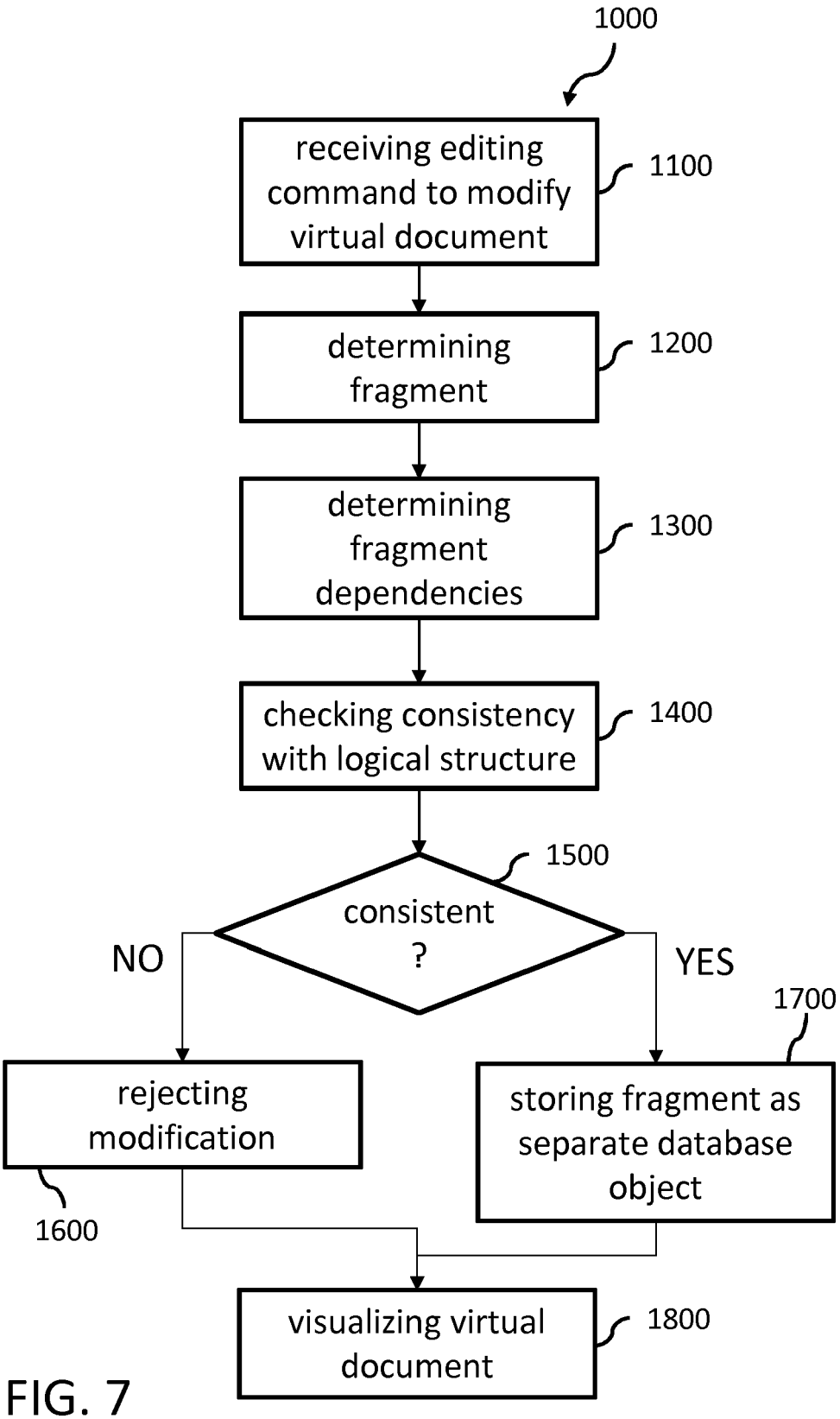

1000
receiving editing command to modify virtual document
1100
determining fragment
1200
determining fragment dependencies
1300
checking consistency with logical structure
1400
consistent ?
1500
NO
YES
rejecting modification
1600
storing fragment as separate database object
1700
visualizing virtual document
1800

FIG. 7

headline of first indenture level to be shown in the table of contents without numbering

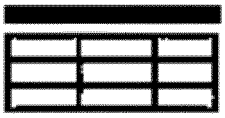
1. numbered headline of first indenture level

text of first indenture level

1.1. numbered headline of second indenture level

text of second indenture level

headline of second indenture level to be shown in the table of contents without numbering



1.2. numbered headline of second indenture level

text of second indenture level

2. numbered headline of first indenture level

FIG. 8

190

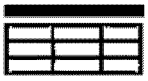
ID	belongs to	type	ind. level	is headline <> numbering	content
001		text	1	without	headline of first indenture level to be shown in the table of contents without numbering
002		text	1	with	numbered headline of first indenture level
003	002	text	2		text of second indenture level
004	002	text	2	with	numbered headline of second indenture level
005	004	text	3		text of third indenture level
006	004	image	3		
007	004	text	3	without	headline of third indenture level to be shown in the table of contents without numbering
008	004	table	3		
009	002	text	2	with	numbered headline of second indenture level
010	009	text	3		text of third indenture level
011	002	text	1	with	numbered headline of first indenture level

FIG. 9

110

Europäisches Patentamt
European Patent Office
Office européen des brevets

EUROPEAN SEARCH REPORT

Application Number
EP 17 15 5254

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (IPC)
Y	GB 2 409 541 A (MANDORLA TECHNOLOGY LTD [GB]) 29 June 2005 (2005-06-29) * page 2, paragraph 5 - page 3, paragraph 3 * * page 12, paragraph 4 - page 13, paragraph 1 * * page 27, paragraph 5 - page 29, paragraph 4; claim 1 *	1,8	INV. G06F17/22 G06F17/24
Y	Phil Fearon: "Merging DITA files in the oXygen Authoring Tool - DeltaXML", , 8 November 2013 (2013-11-08), XP055379936, Retrieved from the Internet: URL:https://www.deltaxml.com/blog/dita/merging-dita-files-in-the-oxygen-authoring-tool/ [retrieved on 2017-06-09] * the whole document *	1,8	
A	US 2008/201365 A1 (PETRI JOHN EDWARD [US]) 21 August 2008 (2008-08-21) * paragraph [0014] * * paragraph [0018] *	1-13	TECHNICAL FIELDS SEARCHED (IPC) G06F
A	EP 1 811 399 A1 (JUSTSYSTEMS CORP [JP]) 25 July 2007 (2007-07-25) * the whole document *	1-13	
A	US 2005/120298 A1 (PETRUJKIC SENJA J [US]) 2 June 2005 (2005-06-02) * the whole document *	1-13	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
Berlin	9 June 2017	Abram, Robert

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 17 15 5254

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

09-06-2017

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
GB 2409541 A	29-06-2005	GB 2409541 A	29-06-2005
		US 2007136662 A1	14-06-2007
		WO 2005062199 A2	07-07-2005
US 2008201365 A1	21-08-2008	NONE	
EP 1811399 A1	25-07-2007	CN 101031912 A	05-09-2007
		EP 1811399 A1	25-07-2007
		US 2008256092 A1	16-10-2008
		WO 2006051904 A1	18-05-2006
US 2005120298 A1	02-06-2005	NONE	

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82